MULTIMEDIA UNIVERSITY ™

STUDENT ID NO | | | | | | | | | |

MULTIMEDIA UNIVERSITY

FIRST TRIMESTER EXAMINATION, **SESSION 2015 / 2016**

TDP3471 – DISTRIBUTED &PARALLEL COMPUTING

(All groups)

15 OCTOBER 2015
2:30 PM – 4:30 PM
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of 7 pages only (including this page).

2. This exam consists of intensive programming questions, so the questions are more in-depth and there are fewer questions than normal exams. Attempt **ALL** questions. The distribution of the marks for each question is given.

3. Print all your answers in the Answer Booklet provided.

4. State all assumptions clearly.

Question 1

You are given 4 dual-core PCs on the same network. We have an 8-stage process where you will receive a stream of data, each piece of data being a character array (string) of 80 characters, and each stage must apply a filter – f0 till f7.

We want a program that uses sockets to process each of these 8 stages on a separate core, passing the result from one core to another, as shown in Figure 1. When the array received is an empty string (i.e. the first character is the null character, '\0'), this indicates that there is no more input.

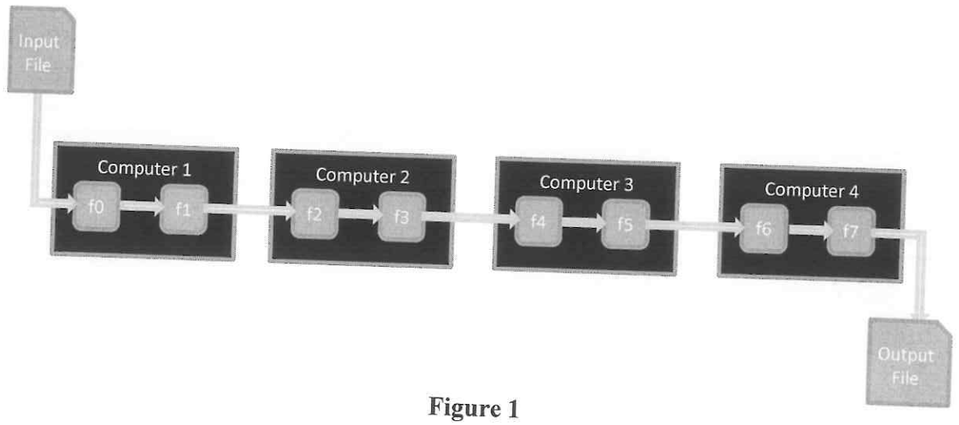

Figure 1

Assuming that the function `void f3(char *)` is given to you, write the code for the f3 program:

- Use socket 5002 for receiving the input from f2 and 5003 for sending the output to f4.
- Assume that Computer 2's IP address is 10.106.20.102 and Computer 3's is 10.106.20.103.
- The program for f3 should act as the host for the program for f2, while the program for f4 should act as the host for the program for f3.
- When the program for f3 receives the terminating condition of empty string, it should send the terminating condition on to the program for f4 before closing its sockets and terminating.

[15 marks]

You may refer to *Appendix 1* for the code from the chat example from tutorial 3.

Question 2

a) Write a serial program to compute [8 marks]

$$\int_{x=0}^{1} \sin(\pi x)\, dx$$

using rectangular quadrature with ten million partitions.

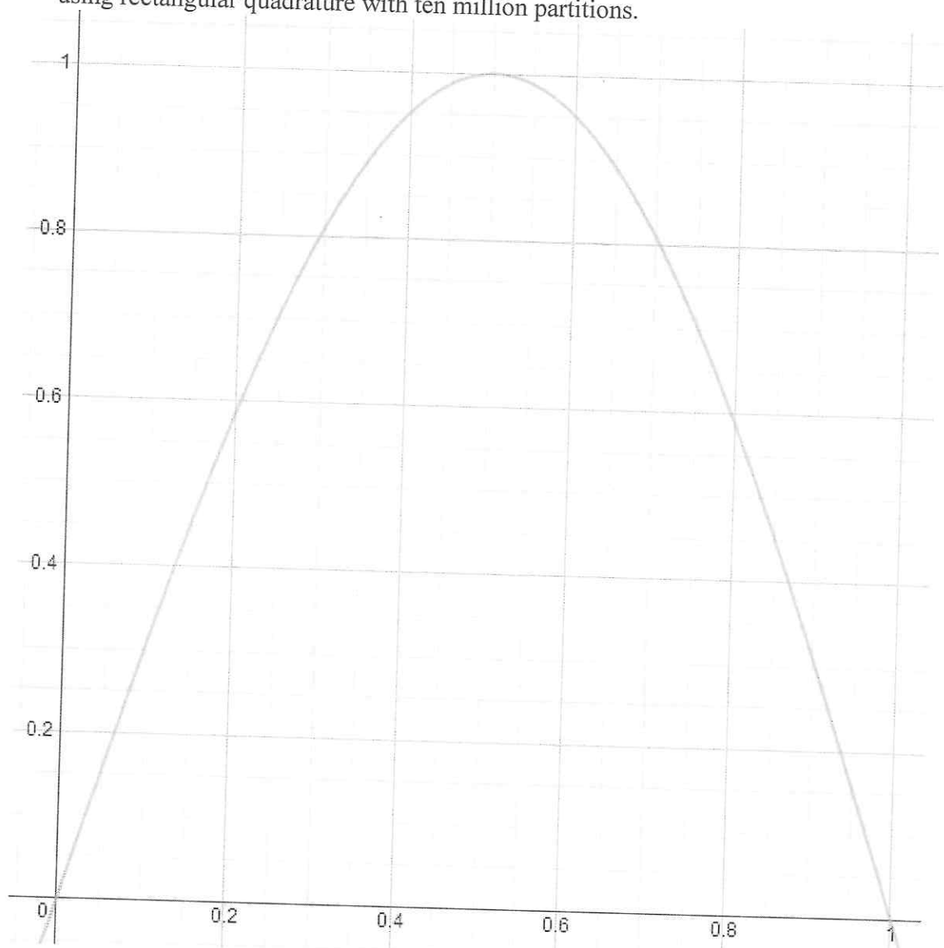

Figure 2: Graph of sin(πx) between 0 and 1

b) Parallelize your serial program of part (a) by using MPI. [7 marks]

You may refer to the function prototypes in *Appendix 2*.

Question 3

The serial program in Listing 3 computes an image based on the long_calculation() function and writes it out as a PGM image file.

```
 1 #include <stdio.h>
 2
 3 #define SIZE 1024
 4
 5 char long_calculation(int,int);
 6
 7 int main(int argc, char *argv[])
 8 { int x, y;
 9   FILE *outfile = fopen("output.pgm","w");    // Open file for writing
10
11   fprintf(outfile,"P5\n%d %d\n255\n",SIZE,SIZE);    // PPM file header
12
13   for (y=0; y<SIZE; y++)
14     for (x=0; x<SIZE; x++)
15       fputc(long_calculation(x,y),outfile);
16
17   fclose(outfile);
18 }
19
20 #include <math.h>
21 #define S 0.02
22
23 char long_calculation(int x, int y)
24 { return ((cos(S*y)*cos(S*x))*128)+127;
25 }
```

Listing 3

Convert this into an OpenMP program so that it would take advantage of a multicore processor. Assume that the function `long_calculation()` is an expensive operations that will take some substantial computation time.

Remember that since the original serial program simply starts writing to the output file in sequential order as soon as one pixel is computed, your OpenMP program must take care of the fact that the computation will now be done concurrently, without losing the advantage of concurrency.

To save on handwriting, you may refer to the line numbers on Listing 3, e.g. "Insert lines 5-7 here." [10 marks]

Appendix 1: chat example from tutorial

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <pthread.h>

#define SOCKET_ERROR        -1
#define BUFFER_SIZE         100
#define QUEUE_SIZE          1
#define PORT                6060

void *listenForMessage(void *arg);
char loop = 1;

int hSocket,hServerSocket;  /* handle to socket */
struct hostent* pHostInfo;   /* holds info about a machine */
struct sockaddr_in Address; /* Internet socket address stuct */
int nAddressSize = sizeof(struct sockaddr_in);
char pBuffer[BUFFER_SIZE];
int nHostPort = PORT;
int n;
pthread_t thread;
pthread_attr_t attr;

int main(int argc, char* argv[])
{ char server = (argc<2);
  char strHostName[255];
  long nHostAddress;

if (server)
{ int optval = 1;
  hServerSocket = socket(AF_INET, SOCK_STREAM, 0);
  setsockopt(hServerSocket, SOL_SOCKET, SO_REUSEADDR, &optval,
    sizeof optval);
  if(hServerSocket == SOCKET_ERROR)
  { printf("Could not make a socket\n");
    return -1;
  }
  /* fill address struct */
  Address.sin_addr.s_addr = INADDR_ANY;
  Address.sin_port = htons(nHostPort);
  Address.sin_family = AF_INET;

  /* bind to a port */
  if(bind(hServerSocket, (struct sockaddr *) &Address, sizeof(Address))
                    == SOCKET_ERROR)
  { printf("Could not bind socket\n");
    return -1;
  }

  /* establish listen queue */
  if(listen(hServerSocket, QUEUE_SIZE) == SOCKET_ERROR)
  { printf("Could not listen\n");
    return -1;
  }

  printf("Server is ready. Start client on the other machine.\n");
  /* get the connected socket */
  hSocket = accept(hServerSocket, (struct sockaddr *)
    &Address,(socklen_t *) &nAddressSize);
  printf("Connected to %s:%d\n", inet_ntoa(Address.sin_addr),
```

```
    ntohs(Address.sin_port));
}
else
{ strcpy(strHostName, argv[1]);
  hSocket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if(hSocket == SOCKET_ERROR)
  { printf("Could not make a socket\n");
    return -1;
  }

  /* get IP address from name */
  pHostInfo = gethostbyname(strHostName);
  if (pHostInfo==NULL)
  { printf("Could not resolve host.\n");
    return -1;
  }
  /* copy address into long */
  memcpy(&nHostAddress, pHostInfo->h_addr, pHostInfo->h_length);

  /* fill address struct */
  Address.sin_addr.s_addr = nHostAddress;
  Address.sin_port = htons(nHostPort);
  Address.sin_family = AF_INET;

  /* connect to host */
  if(connect(hSocket, (struct sockaddr*) &Address, sizeof(Address))
    == SOCKET_ERROR)
  { printf("Could not connect to host\n");
    return -1;
  }
  printf("Connected to %s on port %d\n", strHostName, nHostPort);
}
```

```
/* initialize and set the thread attributes */
pthread_attr_init( &attr );
pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );
pthread_attr_setscope( &attr, PTHREAD_SCOPE_SYSTEM );

n=pthread_create( &thread, &attr, listenForMessage, NULL );

while(loop)
{ printf("Enter your message or 'exit' to end.\n");
  fgets(pBuffer, BUFFER_SIZE, stdin);
  loop=strcasecmp(pBuffer,"exit\n");
  write(hSocket, pBuffer, strlen(pBuffer)+1);
}
/* close socket */
if(close(hSocket) == SOCKET_ERROR)
{ printf("\nCould not close socket\n");
  return -1;
}
printf("End of chat.\n");
return 0;
}

void *listenForMessage(void *arg)
{ while(loop)
  { n = read(hSocket, pBuffer, BUFFER_SIZE);
    pBuffer[n]=0;
    printf("Received message: %s\n", pBuffer);
    loop=strcasecmp(pBuffer,"exit\n");
  }
  exit(0);
}
```


Continued...

Appendix 2: Selected MPI-related functions

double MPI_Wtime()
int MPI_Accumulate(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank,
MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op,
MPI_Comm comm)
int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Barrier(MPI_Comm comm)
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Finalize()
int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
MPI_Datatype recvtype, int root, MPI_Comm comm)
int MPI_Init(int *argc, char ***argv)
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
MPI_Status *status)
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root,
MPI_Comm comm)
int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
MPI_Datatype recvtype, int root, MPI_Comm comm)
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Appendix 3: Selected OpenMP directives

#pragma omp parallel
clause: if *[scalar-expression]*
run_threads *[integer expression]*
default (shared | none)
private (*list*)
firstpriavate (*list*)
shared (*list*)
copyin (*list*)
reduction (*operator*:*list*)
#pragma omp for
clause: private (*list*)
firstpriavate (*list*)
lastpriavate (*list*)
reduction (*operator*:*list*)
schedule (*kind[, chunk_size]*)
collapse (*n*)
ordered
nowait
#pragma omp parallel for